Compilers

Dr. Sherin ElGokhy

Lecture#7

Top-Down Parsing

Outline

- Predictive Parsing
- First Sets
- Follow Sets
- LL(1) Parsing Tables

Predictive Parsers

- Like recursive-descent but parser can "predict" which production to use
  - By looking at the next few tokens
  - No backtracking
- Predictive parsers accept LL(k) grammars
  - L means "left-to-right" scan of input
  - L means "leftmost derivation"
  - k means "predict based on k tokens of lookahead"
  - In practice, LL(1) is used

LL(1) vs. Recursive Descent

- In recursive-descent,
  - At each step, many choices of production to use
  - Backtracking used to undo bad choices
- In LL(1),
  - At each step, only one choice of production
  - That is
    - When a non-terminal A is leftmost in a derivation
    - The next input symbol is t
    - There is a unique production $A \rightarrow \alpha$ to use
      - Or no production to use (an error state)
- LL(1) is a recursive descent variant without backtracking

Predictive Parsing and Left Factoring

- Recall the grammar

  $E \rightarrow T + E \mid T$

  $T \rightarrow int \mid int * T \mid (E)$

- Hard to predict because
  - For T two productions start with int
  - For E it is not clear how to predict

- It is not easy to predict which production to use based on only a single token
- This grammar is not LL(1)
- We need to left-factor the grammar (Rewrite the grammar)

Left-Factoring Example

- Recall the grammar

$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{int} \mid \text{int} * T \mid (E)$$

- *Factor out common prefixes of productions into a single production*
- *Introduce a new non-terminal for the suffixes of the productions that are left factored and write them as alternatives.*
- *Then, it will be multiple productions, one for each suffix.*

$$E \rightarrow T\ X$$

$$X \rightarrow + E \mid \varepsilon$$

$$T \rightarrow (E) \mid \text{int}\ Y$$

$$Y \rightarrow * T \mid \varepsilon$$

Quiz

Choose the alternative that correctly left factors "if" statements in the given grammar

EXPR → if BOOL then { EXPR }
| if BOOL then { EXPR } else { EXPR }
| ...
BOOL → true | false

○ EXPR → if true then { EXPR }
| if false then { EXPR }
| if true then { EXPR } else { EXPR }
| if false then { EXPR } else { EXPR }
| ...

○ EXPR → EXPR' | EXPR' else { EXPR }
EXPR' → if BOOL then { EXPR }
| ...
BOOL → true | false

○ EXPR → if BOOL EXPR'
| ...
EXPR' → then { EXPR }
| then { EXPR } else { EXPR }
BOOL → true | false

○ EXPR → if BOOL then { EXPR } EXPR'
| ...
EXPR' → else { EXPR } | ε
BOOL → true | false

LL(1) Parsing Table Example

- Left-factored grammar

  $E \rightarrow T\,X$ $\qquad$ $X \rightarrow +\,E \mid \varepsilon$

  $T \rightarrow (\,E\,) \mid \text{int}\,Y$ $\qquad$ $Y \rightarrow *\,T \mid \varepsilon$

- The LL(1) parsing table:

next input token

	int	*	+	(	)	$
E						
X						
T						
Y						

leftmost non-terminal

rhs of production to use

LL(1) Parsing Table Example (Cont.)

- Consider the [E, int] entry
  - "When current non-terminal is E and next input is int, use production $E \rightarrow T\ X$"
  - This can generate an int in the first position
- Consider the [Y,+] entry
  - "When current non-terminal is Y and current token is +, get rid of Y"
  - Y can be followed by + only if $Y \rightarrow \varepsilon$

LL(1) Parsing Tables. Errors

- Blank entries indicate error situations
- Consider the [E,*] entry
  - "There is no way to derive a string starting with * from non-terminal E"

LL(1) Parsing Table Example

- Left-factored grammar

  $E \rightarrow T\ X$ $\qquad$ $X \rightarrow +\ E \mid \varepsilon$

  $T \rightarrow (\ E\ ) \mid int\ Y$ $\qquad$ $Y \rightarrow *\ T \mid \varepsilon$

- The LL(1) parsing table:

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Using Parsing Tables

- Method similar to recursive descent, except
  - For the leftmost non-terminal S
  - We look at the next input token a
  - And choose the production shown at [S,a]
- A stack records frontier of parse tree
  - Non-terminals that have yet to be expanded
  - Terminals that have yet to matched against the input
  - Top of stack = leftmost pending terminal or non-terminal
- Reject on reaching error state
- Accept on end of input & empty stack

LL(1) Parsing Algorithm

```
initialize stack = <S $> and next
repeat
   case stack of
      <X, rest>  : if T[X,*next] = Y1…Yn
                      then stack ← <Y1… Yn
  rest>;
                      else  error ();
      <t, rest>   : if t == *next ++
                      then  stack ← <rest>;
                      else error ();
until stack == < >
```

LL(1) Parsing Example

Stack	Input	Action
E $	int * int $	T X
T X $	int * int $	int Y
int Y X $	int * int $	terminal
Y X $	* int $	* T
* T X $	* int $	terminal
T X $	int $	int Y
int Y X $	int $	terminal
Y X $	$	ε
X $	$	ε
$	$	ACCEPT

Quiz

Choose the next parse state given the grammar, parse table, and current state below. The initial string is:

if true then { true } else { if false then { false } } $

	if	then	else	{	}	true	false	$
E	if B then { E } E'				ε	B	B	ε
E'			else { E }		ε			ε
B						true	false	

	Stack	Input
Current	E' $	else { if false then { false } } $
○	$	$
○	else { E } $	else { if false then { false } } $
○	E } $	if false then { false } } $
○	else {if B then { E } E' } $	else { if false then { false } } $

E → if B then { E } E'
| B | ε
E' → else { E } | ε
B → true | false

Computing First Sets

Definition

$$First(X) = \{ t \mid X \rightarrow^* t\alpha \} \cup \{ \varepsilon \mid X \rightarrow^* \varepsilon \}$$

Algorithm sketch:

1. $First(t) = \{ t \}$
2. $\varepsilon \in First(X)$
   - if $X \rightarrow \varepsilon$
   - if $X \rightarrow A_1 \ldots A_n$ and $\varepsilon \in First(A_i)$ for $1 \leq i \leq n$
3. $First(\alpha) \subseteq First(X)$ if $X \rightarrow A_1 \ldots A_n \alpha$ and $\varepsilon \in First(A_i)$ for $1 \leq i \leq n$

First Sets. Example

- Recall the grammar

$E \rightarrow T\,X$ $\qquad$ $X \rightarrow + E \mid \varepsilon$

$T \rightarrow (E) \mid \text{int}\,Y$ $\qquad$ $Y \rightarrow * T \mid \varepsilon$

- First sets

First(( ) = { (} $\qquad$ First(T) = {int, (}

First( )) = {) } $\qquad$ First(E) = {int, (}

First(int) = { int } $\qquad$ First(X) = {+, ε }

First(+) = { + } $\qquad$ First(Y) = {*, ε }

First(*) = { * }

Computing Follow Sets

- Definition:

  $Follow(X) = \{ t \mid S \rightarrow^* \beta X t \delta \}$

- Intuition
  - If $X \rightarrow A B$ then $First(B) \subseteq Follow(A)$ and $Follow(X) \subseteq Follow(B)$
    - if $B \rightarrow^* \varepsilon$ then $Follow(X) \subseteq Follow(A)$
  - If S is the start symbol then $\$ \in Follow(S)$

Computing Follow Sets (Cont.)

Algorithm sketch:

1. $ ∈ Follow(S)
2. First(β) - {ε} ⊆ Follow(X)
   - For each production $A \rightarrow \alpha X \beta$
3. Follow(A) ⊆ Follow(X)
   - For each production $A \rightarrow \alpha X \beta$ where $\varepsilon \in First(\beta)$

Follow Sets. Example

- Recall the grammar

$E \rightarrow T\,X$ $\quad\quad$ $X \rightarrow + E \mid \varepsilon$

$T \rightarrow (E) \mid \text{int}\,Y$ $\quad\quad$ $Y \rightarrow * T \mid \varepsilon$

- Follow sets

Follow(+) = { int, (} Follow(*) = { int, (}

Follow(( ) = { int, (} Follow(E) = {), $}

Follow(X) = {$,) } Follow(T) = {+,) , $}

Follow( )) = {+,) , $} Follow(Y) = {+,) , $}

Follow(int) = {*, +,) , $}

Constructing Parsing Tables: The Intuition

- Consider non-terminal A, production $A \rightarrow \alpha$, & token t
- $T[A,t] = \alpha$ in two cases:

- If $\alpha \rightarrow^* t\ \beta$
  - α can derive a t in the first position
  - We say that $t \in First(\alpha)$

- If $A \rightarrow \alpha$ and $\alpha \rightarrow^* \varepsilon$ and $S \rightarrow^* \beta\ A\ t\ \delta$
  - Useful if stack has A, input is t, and A cannot derive t
  - In this case only option is to get rid of A (by deriving ε)
    - Can work only if t can follow A in at least one derivation
  - We say $t \in Follow(A)$

Constructing LL(1) Parsing Tables

- Construct a parsing table T for CFG G
- For each production $A \rightarrow \alpha$ in G do:
  - For each terminal $t \in First(\alpha)$ do
    $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$, for each $t \in Follow(A)$ do
    $T[A, t] = \alpha$
  - If $\varepsilon \in First(\alpha)$ and $\$ \in Follow(A)$ do
    $T[A, \$] = \alpha$

Notes on LL(1) Parsing Tables

- If any entry is multiply defined then G is not LL(1)
- This means:
  - If G is ambiguous
  - If G is left recursive
  - If G is not left-factored
  - And in other cases as well
- Most programming language CFGs are not LL(1)

Thanks